SLBN: A Scalable Max-min Fair Algorithm for Rate-Based Explicit Congestion Control

Alberto Mozo Jose Luis López-Presa Antonio Fernández Anta

***Abstract*—The growth of the Internet has increased the need for scalable congestion control mechanisms in high speed networks. In this context, we propose a *rate-based explicit congestion control* mechanism with which the sources are provided with the rate at which they can transmit. These rates are computed with a distributed max-min fair algorithm, SLBN. The novelty of SLBN is that it combines two interesting features not simultaneously present in existing proposals: *scalability* and *fast convergence* to the max-min fair rates, even under high session churn. SLBN is scalable because routers only maintain a constant amount of state information (only three integer variables per link) and only incur a constant amount of computation per protocol packet, independently of the number of sessions that cross the router. Additionally, SLBN does not require processing any data packet, and it converges independently of sessions' RTT. Finally, by design, the protocol is conservative when assigning rates, even in the presence of high churn, which helps preventing link overshoots in transient periods. We claim that, with all these features, our mechanism is a good candidate to be used in real deployments.**

I. Introduction

The goal of congestion control algorithms is to fairly share the bandwidth of network links among user flows. The growth of the Internet has increased the need for scalable congestion control mechanisms in high speed networks. For example, it was observed that the current TCP version suffers from performance degradation as the bandwidth-delay product increases [10], [19], [16]. This is mostly due to the slow convergence caused by the AIMD and slow-start mechanisms of TCP. The proposals to face this issue have mainly followed two paths. First, modified versions of TCP, which rely on packet drops or ECN bits, were proposed to improve performance while maintaining scalability (e.g., [10], [11], [19], [22]). Second, new explicit congestion control approaches, based on closed-control-loops, weren proposed (e.g., [9], [14], [16], [23]). These latter approaches provide the sources with explicit feedback about the congestion level of the network. The feedback sent by the routers to the sources is usually an explicit window size or an explicit sending rate. The goal of this explicit feedback is reaching close to 100% link utilization, avoiding packet loss, and achieving quick convergence to a (typically max-min) fair sharing of the links' bandwidths. These approaches are claimed to achieve fast convergence and fair distribution of network resources among sessions. Unfortunately, we have experimentally observed that, even under moderate session churn, these mechanisms do not converge quickly to the max-min fair values. Moreover, they tend to oscillate around the max-min fair rates, causing link overshoots during transient periods. This results in packets being discarded and retransmitted, and, in the end, congestion at the network links. An alternative to these approaches is using explicit end-to-end rated-based flow-control (EERC) protocols like those developed for ATM networks. However, existing EERC proposals (see, e.g., [7], [15]) need to store per-session information in each router link, precluding their deployment in a typical Internet scenario with hundreds of thousands of sessions crossing the backbone routers.

In this work we propose a *rate-based explicit congestion control* mechanism which provides the sources with the explicit rates at which they can transmit. These rates are computed with a distributed algorithm that we call SLBN (from *StateLess B-Neck*). SLBN is, in essence, an EERC protocol. However, it combines two interesting features, not simultaneously present in any other existing mechanism: *scalability* and *fast convergence* to the max-min fair rates. Scalability is achieved by only storing three integer variables per router link, independently of the number of crossing sessions. This clearly differentiates SLBN from EERC mechanisms. Additionally, SLBN keeps the deviations of the rate assignments from their max-min fair values and the convergence speed to the max-min fair rates in the range of the best existing mechanisms, even under high session churn and arbitrary RTTs. They are similar to those of distributed max-min fair algorithms that store per-session information, and much better than those based on closed-control-loops.

A. Related Work

Max-min fairness [5] attempts to allocate the same bandwidth to all contending sessions at each link so that, if a session can not use its bandwidth because of constraints elsewhere in its path, then the residual bandwidth is evenly distributed among the other sessions. Many max-min fair algorithms have been proposed, both centralized and distributed. For scalability, only distributed algorithms can be applied to implement explicit rate congestion control in high speed networks. Gallager [12] and Katevenis [17] achieved fairness by using, at each link, one queue per session and a round robin scheduler, avoiding explicitly computing the rates. With the advent of ATM, many EERC distributed algorithms were proposed to compute virtual circuit max-min

This research was supported in part by the Comunidad de Madrid grant S2009TIC-1692, Spanish MICINN grant TEC2011-29688-C02-01, and National Natural Science Foundation of China grant 61020106002.

fair rates in the Available Bit Rate (ABR) traffic mode (see, e.g. [7], [15], [4], [6], [13]). These algorithms exhibit good convergence speed, that depends linearly on the number of bottlenecks of the network. (Informally, bottlenecks are links that limit the sessions' rates.) Recently, a distributed max-min fair algorithm was proposed, which has the property of stopping all control traffic after computing the rates [20]. All these algorithms require that routers store and process per-session state information at each link to compute the rates, what significantly limits their scalability when hundreds of thousands of session flows cross router links. On the contrary, SLBN only needs to store three integer variables at each router link to compute the rates, independently of the number of flows crossing the link.

Cobb et al. [8] proposed a distributed max-min fair algorithm that does not need per-session information at the links. However, this algorithm depends on a predefined constant parameter T that must be greater than protocol packet RTT. However, it is not easy to upper bound this value because protocol and data packets share network links, and RTT will grow when congestion problems appear. Moreover, we have run experiments in some non-trivial scenarios for which the algorithm does not always converge. Other reduced-state algorithms compute only *approximations* of the rates [3], [2], what can lead to large deviations from the max-min fair rates, since they are sensitive to small changes [1].

An alternative approach to attempt converging to the max-min fair rates is using algorithms based on closed-control-loops. In this case, it is not required to process, classify or store per-session information when a packet arrives to the router. Thus, scalability is not compromised when a large number of sessions cross a router link. For instance, XCP [16] was designed to work well in networks with large bandwidth-delay products. It computes, at each link, window changes which are provided to the source. However, it was shown in [9] that XCP convergence speed can be very slow, and short time duration flows could finish without reaching their fair rate. RCP [9] explicitly computes the rates sent to the sources, what yields more accurate congestion information. Additionally, the computation effort needed in router links per arriving packet is significantly smaller than in the case of XCP. However, in [14] it was shown that RCP does not always converge, and that it does not properly cope with a large number of session arrivals. The authors of [14] propose PIQI-RCP as an alternative to RCP, trying to alleviate the above drawbacks by being more careful when computing the session rates, considering recent rate assignments history. Unfortunately, all these proposals require processing each data packet at each router link to estimate the fair rates, what hampers scalability. Moreover, we have experimentally observed that they often take very long, or even fail, to reach a steady state, when the network topology is not trivial. Additionally, they tend to generate significant oscillations around the max-min fair rates during transient periods, causing link overshoots. A link overshoot scenario implies, sooner or later, a growing number of packets that will be discarded and retransmitted and, in the end, the appearance of congestion problems. All these problems are mainly due to the fact that, unlike they implicitly assume, data from different sessions arrive at different times (due to different and variable RTT), and hence, the rates (based on the estimation of the number of sessions crossing each link) are computed with data which is not synchronously updated.

B. Contributions

In this paper we achieve congestion control by providing each session source with an explicit rate that it can use to limit the traffic it injects in the network. This requires max-min fair rates to be explicitly computed. The cornerstone of our approach is the algorithm that we call SLBN, which computes a rate for each session in real-time. These rates converge very quickly to their max-min fair values, even in the presence of sessions joining and leaving the network. Since current core routers must cope with packets from hundreds of thousands of different session flows, in order to achieve scalability, it is desirable to minimize the processing time and the storage required at the routers. SLBN does not require processing any data packet, and the RTT values of the sessions do not affect its convergence (unlike above mentioned control-loop-based protocols). Additionally, it is scalable because routers only maintain a small amount of state information (only three integer variables per link) and only incur a constant amount of computation per protocol packet, independently of the number of sessions that cross the router. This is mainly achieved by moving the per-session state (w.r.t. the above mentioned max-min fair algorithms, and in particular to the one proposed in [20]) from the routers to the session sources and protocol packets.

We claim that SLBN is suitable for realistic deployment for several reasons. First, its scalability in terms of space and computation, independent of the number of sessions, allows at least thousands of sessions crossing each link. Second, as evaluated by simulation, transient rates are very close to their max-min fair values at every moment. Note that, in a real deployment, the network will be almost permanently in a transient state, because the joining and leaving of sessions will never stop. Hence, transient values should be as accurate as possible if the access routers must control session traffic. Finally, by design, SLBN is conservative when assigning rates, even in the presence of high churn, what helps reducing link overshoots in transient periods.

C. Structure of the Rest of the Paper

Section II describes the system model. In Section III, the proposed algorithm SLBN is described and formally specified. In Section IV, the experiments carried out to evaluate the practical performance of the algorithm are presented. Finally, we present our conclusions in Section V.

II. Definitions and System Model

We have a network composed by routers and hosts which have links connecting them. The network links may have different propagation delays and different bandwidths. Each host is connected to only one router via a dedicated link. Sessions follow a static path in the network. This path starts in a host called the *source*, and ends in another host called the *destination*. The intermediate nodes in the session's path are routers. In our model, each host can only be the source node of one session. (This limitation is just for the sake of simplicity.) When a protocol packet follows the session's path from the source to the destination, we say that it is sent *downstream*. When it follows the session's path from the destination to the source, we say that it is sent *upstream*.

A *rate-based explicit congestion control* mechanism is designed, in which the sources are provided with the explicit rate at which they can transmit, using a max-min fair policy to share links' bandwidth among sessions. Sessions are allowed to specify the maximum rate they need, and to change it dynamically. They are considered greedy in this context, i.e., they want to maximize their assigned bandwidth up to their requested maximum rate. Initially, it can be assumed that the whole bandwidth assigned to a session can be used for session data (considering that control packets use additional bandwidth capacity). However, being more realistic, later in the experiments, it is assumed that data and control packets from a session share the assigned bandwidth, limiting the control traffic to, approximately, 1% of the total.

Knowing when a session is active and when it is no longer active (in order to allocate and deallocate resources to it, the so-called use-it-or-lose-it principle) is key to the success of a rate control mechanism. Therefore, to avoid explicit signaling of session arrivals and departures, some of the max-min fair algorithms mentioned in section I-A estimate the number of sessions that cross a link to subsequently compute the bandwidth that must be assigned to each session. However, as previously noted, the max-min fair problem is very sensitive to small errors in one link, what can produce large errors in some other link [1]. Hence, estimating the number of sessions may generate important oscillations in rate assignments. When these oscillations become permanent, they will eventually yield serious congestion problems, as it will be shown in our experiments.

To avoid these problems, we propose a mechanism where the source nodes participate in an active way, explicitly signaling the arrival and departure of sessions, providing for an exact computation of the number of sessions in the network. The sessions interact with the protocol by means of a set of primitives: A) $API.Join(s, r)$: Session s joins the system and requests a maximum rate r, B) $API.Leave(s)$: Session s is no longer active, C) $API.Change(s, r)$: Session s requests a new maximum rate r, and D)$API.Rate(s, r)$: Used by the protocol to indicate that session s has been assigned a rate r.

From a user point of view, this model could be seen as unrealistic because flows do not know if they are active or not. Our proposal is to take hosts away from the model, and to delegate in the first router (in the session path) the responsibility of implementing the above primitives. As it is commonly assumed (e.g. [25], [18]), access routers maintain information about each individual flow, while core routers, for scalability purposes, do not. Hence, explicitly signaling the arrival and departure of sessions does not compromise the scalability of the access router (e.g., an xDSL home router), since it only has to cope with a small number of sessions. Therefore, it is easy for this kind of routers to execute $API.Join$ when they detect a new flow, or $API.Leave$ when an existing flow times out. Stream oriented flows (e.g., TCP) explicitly signal connection establishment and termination. Hence, these routers can execute the corresponding primitives when they detect the corresponding packets (e.g., SYN and FIN in TCP). On their hand, datagram oriented flows (e.g., UDP) can be tracked down with the help of an array of active flows (e.g. identified by source-destination pairs), and an inactivity timer for each flow. Additionally, the source router can measure in real time differences between the assigned and the actual bandwidth used by a session, and execute $API.Change$ if the actual rate is significantly lower than the rate assigned by the control congestion mechanism.

Reliable communication channels are assumed to be available to transmit SLBN packets. If reliability is not guaranteed, classical techniques can be used to cope with communication errors keeping consistent the state in nodes.

III. Algorithm SLBN

As previously mentioned, SLBN is, in essence, an EERC protocol in which the state information has been moved from the routers to the sources and, the protocol packets. It relies on the session sources performing periodic $Probe$ cycles to discover their max-min fair rates, and to adjust these rates to changes in the network. A $Probe$ cycle starts with the source sending a $Join$ (the first time) or $Probe$ packet downstream. When this packet reaches the destination, it is sent back upstream as a $ProbeAck$ packet which is processed and dropped at the source. At the end of a $Probe$ cycle, the source receives an indication of the maximum amount of bandwidth that can be allocated to the session at every link in its path. When a session leaves the network, a $Leave$ packet is sent downstream, which is dropped at the destination. When a session changes its bandwidth requirements, the new bandwidth is stored, and used in the next $Probe$ cycle.

To simplify the protocol specification, assume that the bandwidth of the first link is always bigger than the rate requested by a session. (Otherwise, add a virtual link with infinite capacity before the actual first link.) The state information (per session) used by SLBN consists of two variables which are kept at the sources, three other variables kept at the routers, and the fields of the protocol packets. At the

```
1  task SourceNode(s, e)
2  var D; pendLeave
3
4  when API.Join(s, r) do
5    D ← r; pendLeave ← FALSE;
6    send downstream Join(s, 0, 0, D, ∅, −1)
7  end when
8
9  when API.Leave(s) do
10   pendLeave ← TRUE
11 end when
12
13 when API.Change(s, r) do
14   D ← r
15 end when
16
17 when received ProbeAck(s, bw'', bw', bw, B, b) do
18   if pendLeave then
19     send downstream Leave(s, bw'', bw', bw, B, b)
20   else
21     API.Rate(s, bw)
22     send downstream Probe(s, bw', bw, min(bw, D), B, b)
23   end if
24 end when
```

Figure 1. Task Source Node of SLBN.

source, variable D stores the rate requested by the session (Figure 1, lines 5 and 14). It will be used in the *Probe* cycles (Figure 1, lines 6 and 22). Another (boolean) variable *pendLeave* indicates that the session is leaving (Figure 1, line 10), so at the end of the current *Probe* cycle, a *Leave* packet will be sent downstream (Figure 1, line 19). Every protocol packet carries the following fields: a) s: The session to which this packet belongs, b) bw'': Bandwidth computed two *Probe* cycles ago, c) bw': Bandwidth computed in the previous *Probe* cycle, d) bw: Bandwidth being computed in this *Probe* cycle, e) B: The set of bottlenecks for this session. Its size in bits is bounded by the diameter of the network, and f) b: The latest bottleneck that was added to B (relative to the path of the session).

During a *Probe* cycle, each router link must identify sessions crossing it as *saturated* or *unsaturated*. A session is identified as saturated if, given the current state at the link, the largest amount of bandwidth that can be allocated to the session at this link is at least that of the field bw of the protocol packets, and it is identified as unsaturated otherwise. The set of saturated sessions is denoted by F, and $BF = \sum_{i \in F} bw_i$ is the total bandwidth allocated to saturated sessions at this link, where bw_i is the bandwidth allocated to session i. The set of unsaturated sessions is denoted by R, and its size by $NR = |R|$. The variables stored at the routers are BF, NR, and N, which is the number of sessions that cross the link.

The largest amount of bandwidth that a router link can offer to a session at a specific moment is called the *equitable shared bandwidth*. Each link computes its equitable shared bandwidth using the following formula: $shBW = \frac{C_e - BF}{NR}$, where C_e is the bandwidth of the link. Therefore, $shBW$ is the maximum bandwidth that the link can allocate to the unsaturated sessions (a.k.a. the *bottleneck level* of the link). A link is a *bottleneck* if it has at least one unsaturated session. At the end of a *Probe* cycle, the source receives the minimum value of $shBW$ for all the links in its path. During transient periods, the values of BF are not stable. Hence, BF could be bigger than C_e. In this case, the value of $shBW$ would be negative, what is not valid. To avoid this possibility, $shBW$ is computed as $shBW = \max\left(\frac{C_e - BF}{NR}, \frac{C_e}{N}\right)$, where $\frac{C_e}{N}$ is a lower bound for the *equitable shared bandwidth*. This limits oscillations in rate assignments, and improves convergence, because the max-min fair rate of a session is always lower-bounded by $\frac{C_e}{N}$ at the most restrictive link.

We claim that scalability at links is guaranteed, since only three integer variables are used, whose length in bits is $O(\log_2(\max(BF, NR, N))$. Therefore, 64 bits for each of them should be enough in practice.

As shown in Figure 1 (line 6), when a session joins the network, a *Probe* cycle starts in which bw'' and bw' are initialized to 0, B is the empty set since there are no bottlenecks discovered yet, and the last discovered bottleneck b is set to -1. When a *Probe* cycle ends, if the session is leaving (marked with the flag *pendLeave*), then a *Leave* packet is sent downstream, with the current values of bw'', bw', bw, B and b. Otherwise, a new *Probe* cycle starts where bw'' is set to bw', bw' is set to bw, and bw is set to $\min(bw, D)$, in order to adjust the requested bandwidth to the actual availability. It is possible to insert a delay between *Probe* cycles to reduce the protocol traffic. In WAN scenarios, RTT values may be large. Therefore, in order to make SLBN highly responsive to changes in the network state, *equitable shared bandwidths* are computed during both the downstream and upstream phases of the *Probe* cycles.

EERC algorithms store the state of each session (whether it is saturated or unsaturated) in the links. However, SLBN keeps this information in the protocol packets, using the fields B (set of bottlenecks) and b (latest bottleneck discovered). To properly compute the *equitable shared bandwidth* ($shBW$) it is necessary to keep variables N, BF and NR in a consistent state. These variables are updated in the following way. N is incremented each time a session joins (Figure 2, line 5), and decremented when it leaves (Figure 2, line 48). NR is incremented each time an unsaturated session joins (Figure 2, line 7), when a session moves from F to R during the downstream phase of a *Probe* cycle (Figure 2, line 16), or during the upstream phase of a *Probe* cycle (Figure 2, line 31). Likewise, it is decremented when a session moves from R to F (Figure 2, lines 22 and 37), and when a session that was in R leaves (Figure 2, line 44).

Keeping BF up to date is a bit more involved. Recall that bw is computed during the *Probe* cycle, so it is not a stable value. Hence, bw' is used to update BF when a session is identified as saturated in the downstream phase of a *Probe* cycle (Figure 2, line 22), or during the upstream phase (Figure 2, line 37). When a session must be moved to

```
1  task RouterLink(e)
2  var BF ← ∅; NR ← 0; N ← 0
3
4  when received Join(s, bw'', bw', bw, B, b) do
5    N ← N + 1; shBW ← max((Ce − BF)/(NR+1), Ce/N)
6    if bw ≥ shBW then
7      B ← B ∪ {e}; b ← e; NR ← NR + 1
8    end if
9    send downstream Join (s, bw'', bw', bw, B, b)
10 end when
11
12 when received Probe(s, bw'', bw', bw, B, b) do
13   if e ∈ B then
14     B ← B \ {e}
15   else
16     BF ← BF − bw''; NR ← NR + 1
17   end if
18   shBW ← max((Ce − BF)/NR, Ce/N)
19   if (e = b) ∨ (bw ≥ shBW) then
20     B ← B ∪ {e}; bw ← shBW; b ← e
21   else // bw < shBW
22     BF ← BF + bw'; NR ← NR − 1
23   end if
24   send downstream Probe (s, bw'', bw', bw, B, b)
25 end when
26
27 when received ProbeAck(s, bw'', bw', bw, B, b) do
28   if e ∈ B then
29     B ← B \ {e}
30   else
31     BF ← BF − bw'; NR ← NR + 1
32   end if
33   shBW ← max((Ce − BF)/NR, Ce/N)
34   if (e = b) ∨ (bw ≥ shBW) then
35     B ← B ∪ {e}; bw ← shBW; b ← e
36   else // bw < shBW
37     BF ← BF + bw'; NR ← NR − 1
38   end if
39   send upstream ProbeAck(s, bw'', bw', bw, B, b)
40 end when
41
42 when received Leave(s, bw'', bw', bw, B, b) do
43   if e ∈ B then
44     NR ← NR − 1
45   else
46     BF ← BF − bw'
47   end if
48   N ← N − 1
49   send downstream Leave(s, bw'', bw', bw, B, b)
50 end when
```

Figure 2. Tasks Router Link of SLBN.

R, BF must be decremented accordingly: in the upstream phase of a *Probe* cycle, since the value added to BF during the downstream phase was bw', that same value is subtracted from BF (Figure 2, line 31). However, during the downstream phase, the value that has to be subtracted from BF is that which was added in the previous *Probe* cycle, i.e. bw'' (Figure 2, line 16). Likewise, when a session that was saturated leaves, the last valid value must be subtracted from BF; in this case, since the value of bw' is not changed at the source (Figure 1, line 19), it is used to update BF (Figure 2, line 46). In the case of a session joining, nothing needs to be done with BF, since bw' and bw'' are 0.

Finally we focus on the update of B and b. Every time $bw \geq shBW$, this link e is added to the set of bottlenecks B of the session, and b is set to e (Figure 2, lines 7, 20, and 35). This is also done when $b = e$ and it has increased its $shBW$ (Figure 2, lines 19 and 34). During both phases of a *Probe* cycle, e is removed from B, until it is verified if it remains being a bottleneck for this session (Figure 2, lines 14 and 29).

The pseudocode for the destination node is not included due to space restrictions. The destination node simply drops *Leave* packets, and bounces *Join* and *Probe* packets back upstream as *ProbeAck* packets.

The convergence of SLBN relies on discovering and stabilizing the bottlenecks of the network. The algorithm starts identifying and stabilizing the most restrictive bottleneck in the network (the one with the smallest bottleneck level). Then, the second most restrictive is identified, and this process continues until all bottlenecks are identified and stabilized. Therefore, in absence of session arrivals or departures in the network, SLBN convergence speed is upper bounded by $O(b)$, where b is the number of distinct bottleneck levels. Now we will give an intuition of how SLBN converges to the max-min fair rates.

Eventually, the set of sessions that cross the link with the most restrictive bottleneck level are notified of their exact rate assignment. Since all $shBW$ values computed by router links are lower bounded by C_e/N_e, the lowest value returned by a *Probe* cycle in the path of the *unsaturated* sessions at the most restrictive link b_1 is $\frac{C_{b_1}}{N_{b_1}}$. (We extend the notation C, N, BF with a subindex to specify to which link they apply.) However, this link could have $BF_{b_1} > 0$, and then its current $shBW$ would be greater than the exact rate assignment. This implies that some session crossing it is restricted in some other link b_i with a lower $shBW$. This situation is not possible, because its $shBW$ is lower bounded by C_{b_i}/N_{b_i} and this value is greater or equal to C_{b_1}/N_{b_1} (recall that b_1 has the most restrictive bottleneck level in the network). Eventually, after some *Probe* cycles, the sessions crossing b_1, currently considered *saturated*, will also be considered *unsaturated* at b_1, and hence, F_{b_1} will become zero. Then, $shBW$ at link b_1 will be equal to $\frac{C_{b_1}}{N_{b_1}}$. Moreover, link b_1 and the sessions bottlenecked at this link become permanently stable, because it is not possible that these sessions can obtain a lower $shBW$ level in a different link.

During subsequent *Probe* cycles, this bottleneck level is propagated to the rest of the links in the path of the sessions bottlenecked at link b_1, which will be identified as *saturated* in the other the links belonging to their paths. These links will recompute their $shBW$ values, and the links with the second most restrictive bottleneck level will be discovered and stabilized in this way. This process continues in the same way in increasing order of bottleneck levels. Thus, all bottlenecks will be eventually identified and stabilized, and all sessions will be stabilized with their max-min fair rates.

IV. Experimental Evaluation

In this section, we experimentally compare the performance of SLBN with other congestion control algorithms. Two of them follow the closed-loop-control approach (namely RCP [9] and PIQI-RCP [14]), two use per-session state at the links (namely Erica [15] and BYZF [4]), and one follows a similar approach as SLBN (namely COBB [8]). The algorithms have been coded in Java and have been run on top of *jmyns*, a home made discrete event simulator. The networks used in the experiments have been generated with the *gt-itm* graph generator, with a typical Internet transit-stub model [24]. Different network topologies and sizes have been considered in the evaluation, but here we only consider WAN scenarios because larger RTT appear in them, and so, the convergence speed of the algorithms is worse and more realistic. Thus, we present the results for two WAN scenarios: one of *Medium* size (1,100 routers) used in the *First Scenario*, and a *Large* one with 11,000 routers for the *Second Scenario*. In both cases 200 hosts are connected to each access router. Link speeds have been set to 100 Mbps in the links connecting hosts and stub routers, to 1 Gbps in the links connecting stub routers, and to 5 Gbps in the links connecting transit routers. The propagation delay of the links has been established as follows: all internal links have been assigned random propagation delays uniformly chosen between 1 and 10 milliseconds, and the links connecting hosts to routers have a 1 microsecond propagation delay. For each session, its source and destination nodes are uniformly chosen at random. The session route is a shortest path between them. In all experiments, the sessions inject data packets, observing their current rate assignment. This way, we can analyze protocol stability when RTT values grow due to rate assignments which are greater than the exact ones.

The parameters we want to analyze in our experiments are: the accuracy of bandwidth assignments in the sources, the accuracy of the bandwidth assignments at the links, the completion time of sessions (when sessions are assigned different fixed amounts of data to transmit), and the stress induced on the link queues. In these experiments, we want to show potential link overshoot problems when high session churn appears in the network. We have used queues of unlimited size, so no packets are dropped at any link. However, if the queues grow, in a real scenario it would imply packet drops, and in the end, congestion problems.

To measure the accuracy of bandwidth assignments at the sources, we look at the distribution of the relative error of the rates at the sources (only for sessions with a computed rate) with respect to their max-min fair values, obtained with a centralized max-min fair algorithm. Let $\lambda_s(t)$ be the rate assigned to session s at time t, and $\lambda_s^*(t)$ the max-min fair assignment given the set of sessions at time t, the error of session s at time t is computed as $E_s(t) = \frac{\lambda_s(t)-\lambda_s^*(t)}{\lambda_s^*(t)} \times 100$. This error reflects the accuracy of the algorithm experienced by the sessions at each point in time. To measure the accuracy of bandwidth assignments at the bottlenecks, we study the relative error of the rates assigned to the sessions that cross a link with respect to the link capacity. Let, $S_e(t)$ be the set of sessions that cross link e at time t, and C_e the bandwidth of link e. Then, the error of a bottleneck e at time t is $E_e(t) = \frac{\sum_{s \in S_e(t)} \lambda_s(t) - C_e}{C_e} \times 100$. This error allows to evaluate the underutilization (or the overshoot) in the bottlenecks, with respect to their maximum capacity.

First Scenario. In this scenario we show that SLBN, being as scalable as closed-loop-control proposals, exhibits a better transient behavior, and in the same range of the best per-session state proposals. Using the *Medium* topology, we have carried out two different experiments[1].

In *Experiment 1a*, 50,000 sessions are started during the first 500 milliseconds, then 10,000 sessions leave the network and 10,000 new sessions join it during the following 200 milliseconds. Figure 3 and Figure 4 show the accuracy of the rate assignments for all the protocols considered, at the sources, and at links. Percentile bars are used to show the deviation from the average. The results show that SLBN, with BYZF and ERICA, assign much more accurate bandwidths than COBB and the algorithms based on closed-control-loops (RCP and PIQI-RCP). In fact, RCP and PIQI-RCP show a quite erratic behavior with errors that are sometimes of 200%. COBB shows a non convergent behavior with average errors of approximately 500%, and values of -15% and 1600% for the 10th and 90th percentiles respectively, during the whole experiment. Therefore, its results are not shown in the figures. Clearly, SLBN and BYZF show the better accuracy, with small deviations from the average, while ERICA tends to make rate assignments a little bit over the exact rate.

In *Experiment 1b*, during 500 milliseconds 50,000 sessions are started. The sessions are assigned fixed amounts of data to be transmitted, following a Pareto distribution with $mean = 300$ packets of 2000 bytes each, and $shape = 4$. Firstly, in this experiment we show, in Figure 6, the evolution of the queues at the links of the network, to state the stress induced on the links. The most notable result is that SLBN is the only one that seems to be able to keep the maximum size of the queues bounded during the whole experiment, and COBB and PIQI-RCP are the ones that impose the biggest stress on the queues. COBB values (not shown in the figure) during the first three seconds are 5293, 8265 and 4775 data packets respectively. Secondly, we show, in Figure 5, the average of the completion times for the sessions of different sizes. SLBN, BYZF, ERICA, PIQI-RCP, and COBB (not shown in the figure) have similar completion times, while RCP has much worse completion times. In the case of PIQI-

[1]The parameters used in RCP experiments are $\alpha = 0.1$ and $\beta = 1$ as suggested in [9]. In PIQI-RCP the parameters used are $T = 5$ ms and the upper bound $\kappa^* = \frac{T}{2(T+RTT)}$ as suggested in Eq. 35 in [14]. In COBB, the parameter used is $T = 70\ ms$ to accommodate to WAN RTTs.

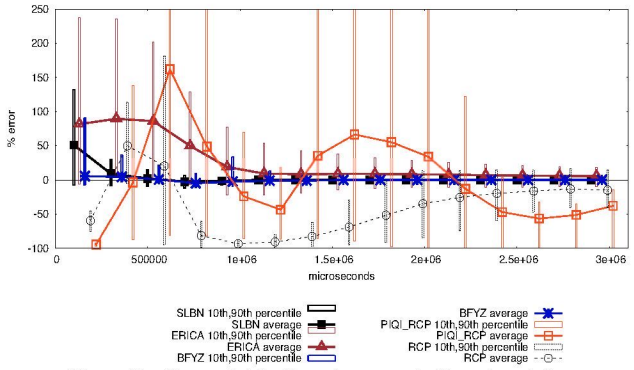

Figure 3. Error distribution at sources in Experiment 1a.

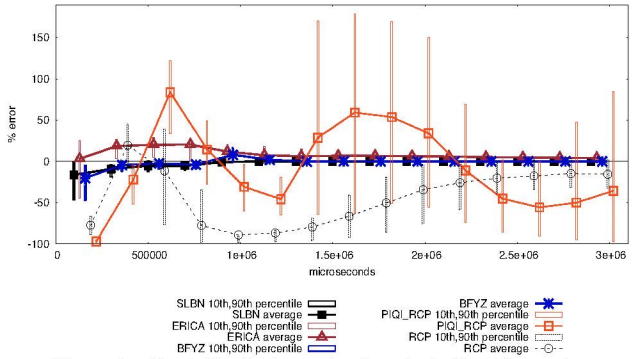

Figure 4. Error distribution at bottlenecks in Experiment 1a.

RCP and COBB, it seems that the queues of unlimited size are assuming the congestion control, improving their results.

Second Scenario. In this scenario, we show that the behavior of SLBN does not depend on the session churn patterns. In this scenario, we use the *Large* network. In this case we compare SLBN only with BFYZ, since the results from the first scenario showed that COBB and the algorithms based on closed-control-loops had a much worse behavior than these two, and ERICA's performance was also worse. In this scenario, we have carried out two experiments.

In *Experiment 2a*, 180,000 sessions are injected during the first second. The results of this experiment are shown in Figures 7 and 8. They show that SLBN performs as well as BFYZ, without the need of per-session information at the routers. Both algorithms quickly reach the max-min fair rates, what implies almost full utilization of the links but never overshooting them significantly. However, SLBN is a bit more conservative than BFYZ. In *Experiment 2b*, 100,000 sessions are injected in the first second. After 9 seconds (when convergence has been reached), 50,000 sessions arrive and 50,000 sessions leave in the next second. The results of this experiment are shown in Figures 9 and 10, where only the second phase of the experiment is shown (the behavior of the algorithms when a massive amount of sessions join the network has been analyzed in Experiment 2a). Again, SLBN and BFYZ behave similarly: there is no significant overshooting at the links (BFYZ's being higher, but always below 5%), and the session churn pattern does not affect the stability of the rates at sources and links.

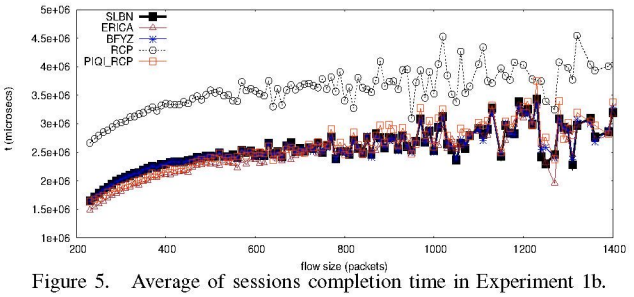

Figure 5. Average of sessions completion time in Experiment 1b.

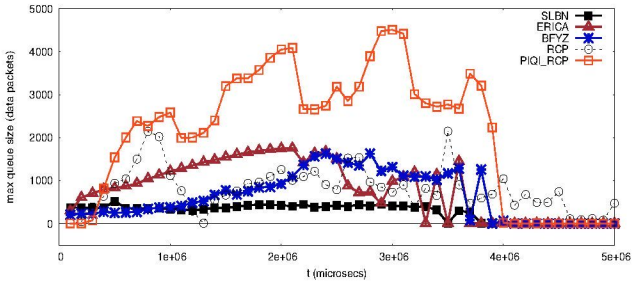

Figure 6. Maximum queue sizes in Experiment 1b.

In summary, we observe in these experiments that SLBN convergence to the max-min fair rates is realized independently of the churn session pattern, with error distributions at sources and links that are nearly constant during transient periods, and achieving a nearly full utilization of links but never overshooting them on average (the maximum of queue sizes is upper bounded by 3% of the link capacity).

V. Conclusions

We can conclude that SLBN is a good candidate to be used as a congestion control mechanism in real deployments in the Internet. SLBN controls congestion by providing each session source with an explicit max-min fair rate that it can use to limit the traffic it injects to the network. The novelty of SLBN is that it combines two interesting features not simultaneously present in existing proposals: *scalability* and *fast convergence* to the max-min fair rates, even under high session churn. SLBN is scalable because routers only maintain a constant amount of state information (only three integer variables per link), and only incur a constant amount of computation per protocol packet, independently of the number of sessions that cross their links. In our experiments, we have observed that, in the case of scalable protocols (RCP and PIQI_RCP), the inaccurate estimation of the number of sessions that cross a link leads to erratic behaviors during transient periods. Therefore, we propose that access routers explicitly signal the arrivals and departures of sessions. This way, SLBN convergence speed is similar to that of the fastest, but non-scalable, EERC protocols. Moreover, as our experiments show, SLBN is conservative when assigning rates, when compared with several well known existing proposals (ERICA, BYZF, COBB, RCP, and PIQI_RCP). As a consequence, no overshoot is produced at router links, and the queue sizes are kept small and almost constant, even under high session churn. Finally, since SLBN control

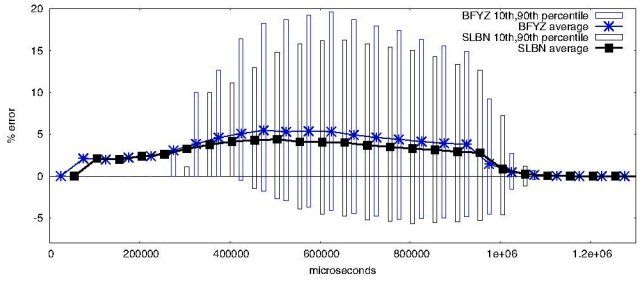

Figure 7. Error distribution at sources in Experiment 2a.

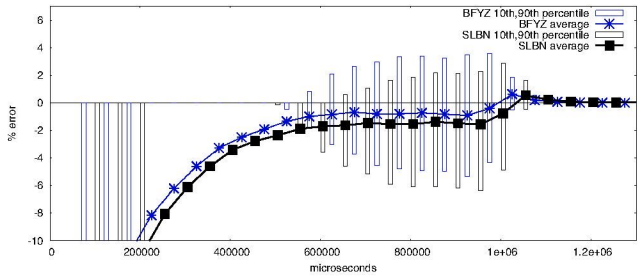

Figure 8. Error distribution at bottlenecks in Experiment 2a.

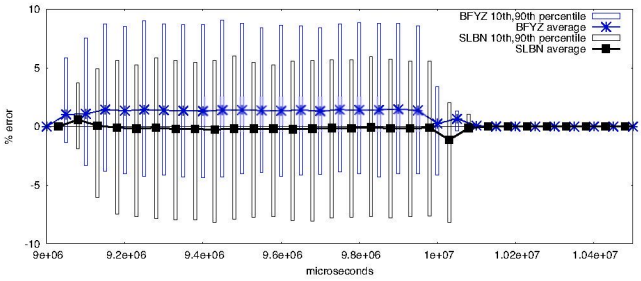

Figure 9. Error distribution at sources in Experiment 2b.

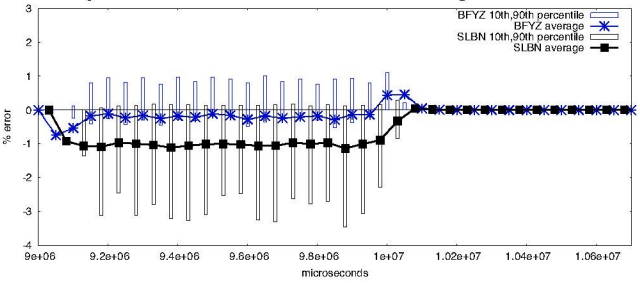

Figure 10. Error distribution at bottlenecks in Experiment 2b.

packets do not suffer significant delays when processed at link queues (due to the small size of queues), rates assigned during transient periods are very close to the optimal.

REFERENCES

[1] Y. Afek, Y. Mansour, and Z. Ostfeld. Convergence complexity of optimistic rate-based flow-control algorithms. *J. Algorithms*, 30(1):106–143, 1999.

[2] Y. Afek, Y. Mansour, and Z. Ostfeld. Phantom: a simple and effective flow control scheme. *Computer Networks*, 32(3):277–305, 2000.

[3] B. Awerbuch and Y. Shavitt. Converging to approximated max-min flow fairness in logarithmic time. In *INFOCOM*, pages 1350–1357, 1998.

[4] Y. Bartal, M. Farach-Colton, S. Yooseph, and L. Zhang. Fast, fair and frugal bandwidth allocation in atm networks. *Algorithmica*, 33(3):272–286, 2002.

[5] D. Bertsekas and R. G. Gallager. *Data Networks (2nd Edition)*. Prentice Hall, 1992.

[6] Z. Cao and E. W. Zegura. Utility max-min: An application-oriented bandwidth allocation scheme. In *INFOCOM*, pag. 793–801, 1999.

[7] A. Charny, D. Clark, and R. Jain. Congestion control with explicit rate indication. In *ICC'95, vol. 3*, pag. 1954 – 1963.

[8] J. A. Cobb and M. G. Gouda. Stabilization of max-min fair networks without per-flow state. In volume 5340 of *Lect. Notes in Comp. Science*, pages 156–172. Springer, 2008.

[9] N. Dukkipati, M. Kobayashi, R. Zhang-Shen, and N. McKeown. Processor sharing flows in the internet. In *IWQoS*, pages 271–285, 2005.

[10] S. Floyd. High-speed TCP for Large Congestion Windows. RFC 3649 (Experimental), Dec. 2003.

[11] S. Ha, I. Rhee, and L. Xu. Cubic: a new tcp-friendly high-speed tcp variant. *SIGOPS Oper. Syst. Rev.*, 42:64–74, 2008.

[12] E. L. Hahne and R. G. Gallager. Round robin scheduling for fair flow control in data communication networks. In *IEEE Int. Conf. in Comm., ICC'86*, pages 103–107, 1986.

[13] Y. T. Hou, H. H.-Y. Tzeng, and S. S. Panwar. A generalized max-min rate allocation policy and its distributed implementation using abr flow control mechanism. In *INFOCOM*, pages 1366–1375, 1998.

[14] S. Jain and D. Loguinov. PIQI-RCP: design and analysis of rate-based explicit congestion control. In *Fifteenth IEEE Int. Work. on QoS, IWQoS 2007*, pages 10 –20, June 2007.

[15] S. Kalyanaraman, R. Jain, S. Fahmy, R. Goyal, and B. Vandalore. The erica switch algorithm for abr traffic management in atm networks. *Networking, IEEE/ACM Transactions on*, 8(1):87–98, 2000.

[16] D. Katabi, M. Handley, and C. E. Rohrs. Congestion control for high bandwidth-delay product networks. In *SIGCOMM*, pages 89–102, 2002.

[17] M. Katevenis. Fast switching and fair control of congested flow in broadband networks. *IEEE Journal on Selected Areas in Communications*, SAC-5(8):1315–1326, 1987.

[18] J. Kaur and H. Vin. Core-stateless guaranteed throughput networks. In *INFOCOM 2003. 22nd Annual Joint Conf. of the IEEE Computer and Communications. IEEE Societies*, vol. 3, pages 2155–2165. IEEE, 2003.

[19] T. Kelly. Scalable TCP: improving performance in high-speed wide area networks. *ACM SIGCOMM Comp. Comm. Rev.*, 33:83–91, 2002.

[20] A. Mozo, J. L. López-Presa, and A. Fernández Anta. B-neck- a distributed and quiescent max-min fair algorithm. In *Proc. 10th IEEE Int. Symp. on Network Computing and Applications (IEEE NCA 2011)*, USA, 2011.

[21] D. Nace and M. Pióro. Max-min fairness and its applications to routing and load-balancing in communication networks: A tutorial. *IEEE Comm. Surv. and Tut.*, 10(1-4):5–17, 2008.

[22] D. X. Wei, C. Jin, S. H. Low, and S. Hegde. Fast tcp: motivation, architecture, algorithms, performance. *IEEE/ACM Trans. Netw.*, 14:1246–1259, December 2006.

[23] Y. Xia, L. Subramanian, I. Stoica, and S. Kalyanaraman. One more bit is enough. *IEEE/ACM Trans. Netw.*, 16:1281–1294, 2008.

[24] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee. How to model an internetwork. In *INFOCOM*, pages 594–602, 1996.

[25] Z. Zhang, Z. Duan, L. Gao, and Y. Hou. Decoupling qos control from core routers: A novel bandwidth broker architecture for scalable support of guaranteed services. In *ACM SIGCOMM Comp. Comm. Review*, vol. 30, pages 71–83, 2000.